Attorney's Docket No.: 15670-075001 / SD2004-151

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: DETECTING PUBLIC NETWORK ATTACKS USING SIGNATURES AND FAST CONTENT ANALYSIS

APPLICANT: GEORGE VARGHESE, SUMEET SINGH, CRISTI ESTAN AND STEFAN SAVAGE

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 399297925 US

April 8, 2004

Date of Deposit

DETECTING PUBLIC NETWORK ATTACKS USING SIGNATURES AND FAST CONTENT ANALYSIS

Statement As To Federally-Sponsored Research

[0001] The U.S. Government has certain rights in this disclosure pursuant to Grant Nos. 60NANB1D0118 and ANI-0137102 awarded by the Com. National Institute of Standards and Technology and National Science Foundation.

Background

[0002] Many computers are connected to publicly-accessible networks such as the Internet. This connection has made it possible to launch large-scale attacks of various kinds against computers connected to the Internet. A *large-scale attack* is an attack that involves several sources and destinations, and which often (but not necessarily) involves a large traffic footprint. Examples of such large-scale attacks may include:

viruses, in which a specified program is caused to run on the computer, which then attempts to spread itself to other computers known to the host computer (e.g., those listed in the address book),

denial of service attacks (DoS), in which a group of computers is exposed to so many requests that it effectively loses the ability to respond to legitimate

requests. Many viruses and worms indirectly cause DoS attacks as well for networks by sending a huge amount of traffic while replicating. *Distributed denial of service (DDOS)* occurs when an attacker uses a group of machines (sometimes known as zombies) to launch a DoS attack. *Backdoor or Vulnerability Scanning*: Another form of large-scale attack is where an intruder scans for *backdoors* at machines or routers. A backdoor is a method by which a previously attacked machine can then be enlisted by future attackers to be part of future attacks.

[0003] *Large-scale spam*: Spam is unsolicited network messages often sent for commercial purposes. Large-scale spam is often simply the same as (or small variants of) the spam sent to multiple recipients. Note that this definition of spam includes both email as well as newer spam variants such as Spam Sent Over Instant Messenger.

[0004] A specific form of attack is an *exploit*, which is a technique for attacking a computer, which then causes the intruder to take control of the target computer, and run the intruder's code on the attack machine. A *worm* is a large-scale attack formed by an *exploit* along with propagation code. Worms can be highly efficacious, since they can allow the number of infected computers to increase geometrically.

[0005] Many current worms propagate via random probing. In the context of the Internet, each of the number of different computers has an IP address, which is a 32-bit address. The probing can simply randomly probe different combinations of 32-bit addresses, looking for machines that are susceptible to the particular worm. Once the machine is infected, that machine starts running the worm code, and again begins the Internet. This geometrically progresses. However, future worms may not use random probing, so probing can only be used as one sign of a worm.

[0006] The worm can do some specific damage, or alternatively can simply take up network bandwidth and computation, or can harvest e-mail addresses or take any other desired action.

[0007] A very common exploit is a so-called *buffer overflow*. In computers, different areas of memory are used to store various pieces of information. One area in memory may be associated with storing information received from the network: such areas are often called buffers. However, an adjoining area in the memory may be associated with an entirely different function. For example, a document name used for accessing Internet content (e.g., a URL) may be stored into a URL buffer. However, this URL buffer may be directly adjacent to protected memory used for program

access. In a buffer overflow exploit, the attacker sends a URL that is longer than the longest possible URL that can be stored in the receiver buffer and so *overflows* the URL which allows the attacker to store the latter portion of its false URL into protected memory. By carefully crafting an extra long URL (or other message field), the attack or can overwrite the return address, and cause execution of specified code by pointing the return address to the newly installed code. This causes the computer to transfer control to what is now the attacker code, which executes the attacker code.

[0008] The above has described one specific exploit (and hence worm) exploiting the buffer overflow. A security patch that is intended for that exact exploit can counteract any worm of this type. However, the operating system code is so complicated that literally every time one security hole is plugged, another is noticed. Further, it often takes days for a patch to be sent by the vendor; worse, because many patches are unreliable and end users may be careless in not applying patches, it may be days, if not months, before a patch is applied. This allows a large window of vulnerabilityduring which a large number of machines are susceptible to the corresponding exploit. Many worms have exploited this window of vulnerability.

[0009] A *signature* is a string of bits in a communication packet that characterize a specific attack. For example, an attempt to execute the perl program at an attacked machine is often signalled by the string "perl.exe" in a message/packet sent by the attacker. Thus a signature-based blocker could remove such traffic by looking for the string "perl.exe" anywhere in the content of a message. The signature could, in general, include header patterns as well as exact bit strings, as well as bit patterns (often called *regular expressions*) which allow more general matches than exact matches.

[0010] While the exact definition of the different terms above may be a matter of debate, the basic premise of these, and other attacks, is the sending of undesired information to a publicly accessible computer, connected to a publicly accessible network, such as the internet.

[0011] Different ways are known to handle such attacks. One such technique involves using the *signature*, and looking for that signature in Internet traffic to block anything that matches that signature. A limitation of this technique has come from the way that such signatures are found. The signature is often not known until the first attacks are underway, at which point it is often too late

to effectively stop the initial(sometimes called zero-day) attacks.

[0012] An *Intrusion Detection System (IDS)* may analyze network traffic patterns to attempt to detect attacks. Typically, IDS systems focus on known attack signatures. Such intrusion detection systems, for example, may be very effective against so-called script kiddies who download known scripts an attempt to use them over again at some later time.

[0013] Existing solutions to attacks each have their own limitations. Hand patching is when security patches from the operating system vendor are manually installed. This is often too slow (takes days to be distributed). It also requires large amounts of resources, e.g., the person who must install the patches.

[0014] A firewall may be positioned at the entrance to a network, and review the packets coming from the public portion of the network. Some firewalls only look at the packet headers; for example, a firewall can route e-mail that is directed to port 25 to a corporate e-mail gateway. The firewalls may be useful, but are less helpful against disguised packets, e.g., those disguised by being sent to other well-known services.

[0015] Intrusion detection and prevention systems, and signature based intrusion systems look for an intrusion in the network. These are often too slow (because of the time required for humans to generate a signature) to be of use in a rapidly spreading new attack.

[0016] Other systems can look for other suspicious behavior, but may not have sufficient context to realize that certain behavior accompanying a new attack is actually suspicious. For example, a common technique is to look for scanning behavior but this is ineffective against worms and viruses that do not scan. This leads to so-called false negatives where more sophisticated attacks (increasingly common) are missed.

[0017] Scanning makes use of the realization that an enterprise network may be assigned a range of IP addresses, and may only use a relatively small portion of this range for the workstations and routers in the network. Any outside attempts to connect to stations within the unused range may be assumed to be suspicious. When multiple attempts are made to access stations within this address space, they may increase the level of suspicion and make it more likely that a scan is taking place.

[0018] This technique has been classically used, as part of the so-called network telescope approach.

Summary

[0019] The present application defines a new technique of automatically determining an unknown attack. This is done by looking for specified different commonalities among the different attacks, and can be used to automatically generate information indicative of the new attack, e.g., a signature for the new attack.

[0020] The embodiments describe detecting commonalities including content replication, increasing levels of sending, scanning, executable code, and/or spam keywords.

[0021] An aspect of this system defines scalable data reduction techniques to detect the commonalities refered to above. These scalable techniques enable the monitoring of network data for any desired size network apparatus that can be implemented at various speeds ranging from the highest speed links (currently at 40 Gbps) to lower speed local links within local networks.

Brief description of the drawings

[0022] These and other aspects will now be described in detail with reference to the accompanying drawings, wherein:

[0023] Figure 1 shows a basic view of the computer on the network with a firewall and an intrusion detection/prevention system;

[0024] Figure 2 shows a block diagram of the system of the present system including the various block;

[0025] Figure 3 shows a data reduction technique including both a scalable solution and a simple solutions;

[0026] Figure 4 shows the scalable solution using a hash solutions;

[0027] Figure 5 shows a technique of obtaining different data portions from network packets to be used as signatures;

[0028] Figures 6a-6d show different systems of detecting increasing numbers of sources and destinations;

[0029] Figure 7 shows a block diagram of the scan test;

[0030] Figures 8a and 8b show the code test operations;

[0031] Figure 9 shows the correlation test;

[0032] Figure 10 shows details of the Spam test;

[0033] Figures 11A and 11B show different forms of the intrusion detection and prevention system configurations.

Detailed description

[0034] Figure 1 shows a basic embodiment of a computer system 100 connected to a publicly available network 110.

While the publicly available network is shown and described herein as being the Internet, it should be understood that this can be used with any network connected to any computer. An entry device 120 sits between the network 110 and the computer 100. The entry device may include the functionalities described throughout this specification.

[0035] Internet messages are sent in packets including headers that identify the destination and/or function of the message. An IP header identifies both source and destination for the payload. A TCP header may also identify destination and source port number. The port number identifies the service which is requested from the TCP destination in one direction, and from the source in the reverse direction. For example, port 25 may be the port number used commonly for e-mail; port number 80 is often used for FTP and the like. The port number thus identifies the specific resources which are requested.

[0036] An intrusion is an attempt by an intruder to investigate or use resources within the network 110 based on messages over the network. A number of different systems are in place to detect and thwart such attacks.

[0037] The inventors have discovered commonalities between the different kinds of large-scale attacks, each of which

attack a different security hole, but each of which have something in common.

[0038] Typical recent attacks have large numbers of attackers. Typical recent attacks often increase geometrically, but in any case the number of infected machines increases.

[0039] Attacks may often be polymorphic, that is they change their content during each infection in order to thwart signature based methods.

[0040] The present technique describes using properties of an attack to detect properties of the new attack, by detecting patterns in data. Effectively, this can detect an attack in the abstract, without actually knowing anything about the details of the attack. The detection of attack can be used to generate a signature, allowing automatic detection of the attack. Another aspect describes certain current properties which are detected, to detect the attack.

[0041] A technique is disclosed which identifies characteristics of an abstract attack. This abstract attack looks for properties in network data which make it likely that an attack of new or previous type is underway.

[0042] The present disclosure describes a number of different properties being viewed; however it should be

understood that these properties could be viewed in any order, and other properties could alternatively be viewed, and that the present disclosure only describes a number of embodiments of different ways of finding an attack under way.

[0043] One aspect of the disclosed technique involves looking through large amounts of data. An aspect discloses a truly brute force method of looking through this data; and this brute force method could be usable if large amounts of resources such as memory and the like are available. Another aspect describes scalable data reduction techniques, in which patterns in the data are determined with reduced resources (specifically, smaller configurations of memory and processing.)

[0044] Figure 2 shows a basic block diagram of an embodiment. Portions of data on a network, e.g., packets 200 are found by examining network messages coming in on a vantage link, which can be, for example, any vantage point which can read data portions from the network. The packet data is analyzed in two parallel analysis blocks. A content checker Part 199 includes parts 205, 215, 230 and 245 in Figure 2. A destination checker Part 190 includes Parts 255 and 265 in Figure 2. Both parts extract anomalous signatures from different criteria. The

destination checker 190 is based on a special assumption that there is known vulnerability in a destination machine. This makes the problem easier and faster. The content checker Part 199 makes no such assumption. The parts can be implemented in parallel to reduce computation time.

[0045] The destination checker 190 analyzes the packets 200 for known vulnerabilities such as buffer overflows. At 255, a list of destinations that are susceptible to known vulnerabilities is first consulted to check whether the destination of the current packet being analyzed is on the list. Such a list can be built by a scan of the network prior to the arrival of any packets containing an attack and/or can be maintained as part of routine network maintenance.) If the specific destination is susceptible to a known vulnerability, then the packet is intended for that destination parsed at 265 to see whether the packet data conforms to the vulnerability. For example, in a buffer overflow vulnerability for say a URL, the URL field is found and its length is checked to see if the field is over a pre-specified limit. If the packet passes the tests in 255 and 265, the relevant contents of the packet that exploit the vulnerability (for example, the contents of the field that would cause a buffer overflow) are passed

to the output as an anomalous signature, together with the destination and source of the packet.

[0046] Content analysis 199 creates anomalous signatures for attacks that are not necessarily based on known vulnerabilities. At 205, "signatures" that are indicative of the information in the packets are created. The signatures can represent a reduced data portion of the information, or a portion or slice of the information, for example. These signatures are analyzed through the techniques of Figure 2 to determine "common content".

[0047] At 215, the signatures 210 are analyzed to determine frequent content within the data itself, as the common content.

[0048] It has been found that large attacks against network resources typically include content which repeats an unusual number of times. For example, the content could be TCP or IP control messages for denial of service attacks. By contrast, worms and viruses have content that contains the code that forms the basis of the attack, and hence that code is often repeated as the attack propagates from computer to computer. Spam has repeated content that contains the information the spammer wishes to send to a large number of recipients.

[0049] Only the frequent signatures are likely to be problems. For example, a signature that repeats just once could not represent a large-scale attack. At most, it represents an attack against a single machine. Therefore, the frequent signatures 225 which have been found at 215 may be further analyzed to determine if it is truly a threat, or is merely part of a more benign message.

[0050] The signatures found to be frequent signatures at 215 are further analyzed to detect spreading content at 230 as the common content. This spreading content test determines whether a large (where "large" is defined by thresholds that can be set to any desired level) number of attackers or attacked machines are involved in sending/receiving the same content. The content is "common," in the sense that the same frequent signatures are being sent. During a large-scale attack, the number of sources or destinations associated with the content may grow geometrically. This is in particular true for worms and viruses. For spam, the number of destinations to which the spam content is sent may be relatively large; at least for large-scale spam. For denial of service attacks, the number of sources may be relatively large. Therefore, spreading content may be an additional factor representing an ongoing attack.

[0051] The frequent and spreading signatures found at 240 may then be subjected to additional checks at 245. These additional checks can check for code, spam, backdoors, *scanning and correlation*. Each of these checks, and/or additional checks, can be carried out by modules, either software based, hardware based, or any combination thereof.

[0052] In order to launch an attack, it may be necessary to communicate with vulnerable sources. *Scanning* may be used to find valid IP addresses to probe for vulnerable services. Probing of unused addresses and/or ports can be used for this determination. However it is possible that future attacks may modify their propagation strategies to use pre-generated addresses instead of probing. Therefore, this invention uses scanning only as an "additional sign of guilt" which is not necessary to output an anomalous signature.

[0053] *Correlation* refers to the fact that a piece of content sent to a set of destinations in a measurement interval is followed, in a later measurement interval, by some fraction of these destinations acting as sources of the content. Such correlation can imply causality wherein infections sent to stations in the earlier interval are followed by these stations acting as infecting agents in the later interval.

[0054] Besides scanning and correlation, the presence of executable code segments is also an "additional sign of guilt". Worms and certain other attacks are often characterized by the presence of code (for example, code that can directly execute on Windows machines) in the attack packets they send. Therefore, in analyzing content to determine an infestation, the repeatable content is tested against parameters that determine executable code segments. It is unlikely that reasonably large segments of contiguous packet data will accidentally look like executable code; this observation is the basis of special techniques for determining code that are described herein with reference to Figs. 8a and 8b. In one aspect, a check is made for Intel 8086 and Unicode executable code formats.

[0055] Spam is characterized by an additional sign of guilt such as keywords based on heuristic criteria.

[0056] Note that while worms may evince themselves by the presence of reasonably large code fragments, other attacks such as Distributed Denial of Service may be based on other characteristics such as large amounts of repetition, large number of sources, and the reception of an unusually large number of TCP reset messages.

[0057] These additional checks 245 may be optional, any number of, all of, or none of these tests may be used.

[0058] The above has described mechanisms for analyzing content to detect attack signatures and sources and destinations. It should be understood, however, that the brute force method of analyzing content (e.g., in 215 and 230) could occupy incredible amounts of data storage. For example, commonly used vantage links that operate at 1 Gigabit per second, easily produce terabytes of packet content over a period of a few hours. Accordingly, a general data reduction technique may be used. It should be understood, however, that in one aspect, the other detection techniques may be used without the general data reduction technique. According to an aspect, a data reduction technique is used as part of the detection at 205, 215, 230 and/or 245.

[0059] In one aspect, anomalous signatures may be established when any frequent content found by 215 meets an additional test found by 230 or 245. According to another aspect, the signatures may be scored based on the amount on indicia they include. In any case, the information is used at 285, to form "anomalous signatures" which may be used to block operations, or may be sent to a bank of signature blockers and managers.

[0060] In addition to the signature, if a packet signature is deemed to be anomalous according to the tests above, the

destination and source of the packet may also be passed (285) to the output. This can be useful, for example, to track which machines in a network have been attacked, and which ones have been infected.

[0061] At 275, the protection device may also (in addition to passing the signature, source, and destination) take control actions by itself. Standard control actions that are well known in the state of the art include connection termination(where the TCP connection containing the suspicious signature is terminated), connection rate limiting (where the TCP connection is not terminated but slowed down to reduce the speed of the attack), packet dropping (where any packet containing the suspicious content is dropped with a certain probability). Note that when the attack is based on a known vulnerability (255, 265) packet dropping with probability 1 can potentially completely prevent an attack from coming into a network or organization.

[0062] ***Signature Computation Block (205):*** The signature computation block is detailed with respect to Fig 5. The signature *S* can simply be any subset of the TCP payload and/or header. Figure 5 shows the different alternatives for the signatures. A general signature can simply be any subset, shown in 500. A specific payload signature can be

formed from the TCP payload added to or appended to the TCP destination (or source) port shown as 505. This signature recognizes that many attacks target specific destination (or in some limited cases, source) ports. An offset signature is based on the recognition that modern large-scale attacks may become *polymorphic* --- that is, may modify the content on individual attack attempts. This is done to make each attack attempt look like a different piece of content. Complete content change is unlikely, however. Some viruses add small changes, while others encrypt the virus but add a decryption routine to the virus. Each contains some common piece of content; in the encryption example, the decryption routine would be the common piece of content.

[0063] The attack content may lie buried within the packet content and may be repeated, but other packet headers may change from attack to attack. Thus, according to another aspect, shown as the offset signature 570, the signature is formed by any continuous portion in payload, appended to the TCP destination port. Therefore, the signature 510 investigates for content repetition strings anywhere within the TCP payload. For example, the text "hi Joe" may occur within packet 1 at offset 100 in a first message, and the same text "hi Joe" may occur in packet 2 at offset 200.

This signature 510 allows counting that as two occurrences of the same string despite the different offsets in each instance.

[0064] The evaluation of this occurrence is carried out by evaluating all possible substrings in the packet of any certain length. A value of a substring length can be chosen, for example, 40 bytes. Then, each piece of data coming in may be windowed, to first look for bytes 1 through 40, then look for bytes 2 through 41, then look for bytes 3 through 42. All possible offsets are evaluated.

[0065] The length of substrings may be selected a trade-off depending on the desired amount of processing. Longer substrings will typically have fewer false positives, since it is unlikely that randomly selected substrings can create repetitions of a larger size. On the other hand, shorter substrings may make it more difficult for an intruder to evade attacks.

[0066] Certain attacks may chop the attack into portions which are separated by random filler. However, this will still find several invariant content substrings within the same packet.

[0067] The multi-signature 515 may be used to combat these kinds of attacks, formed by one or more continuous portions of payload, and the destination port. The multi-signature,

is formed by one or more continuous portions of a packet concatenated with a destination port.

[0068] Find Frequent Signature Block

[0069] After computing a signature 210 at 205, the signature is subjected to the frequent signature test at 215. The find frequent signatures operation 215 is detailed with reference to the flowchart of Figure 3.

[0070] The content repetition can be any pre-specified set of bytes within the packet at any location. This may include a part of the header or the payload or just a portion of the payload.

[0071] The simplest way of looking for this content is to simply store each item within each message in a database. However, this could require a massive amount of storage at the detection device. An embodiment describes scalable data reduction techniques to avoid this excessive memory usage.

[0072] At 300, a data reduction signature is taken of a string *S*, within the received network content. The data reduction takes each item of data, which can be a fixed size or a variable size, and reduces that data to something less, but which reduced data has a constant predetermined relation with the original data. The process must be repeatable, e.g., each time the same original data is

received, the same reduced data will be produced. However, information theory dictates that at least certain other data will also produce the same reduced data.

[0073] A specific data reduction technique which is described herein is *hashing*. Hashing is a set of techniques to convert a long string or number into a smaller number. A simple hashing technique is often to simply remove all but the last three digits of a large number. Since the last three digits of the number are effectively random, it is easy way to characterize something that is referred by a long number. For example, U.S. Patent No. 6,398,311 can be described simply 'the 311 patent'. However, much more complex and sophisticated forms of hashing are known.

[0074] In one example, assume the number 158711, and that this number must be assigned to one of 10 different hashed "bins" by hashing the number to one of 10 bins. One hashing technique simply adds the digits 1 + 5 + 8 + 7 + 1 + 1 equals 23. The number 23 is still bigger than the desired number of 10. Therefore, another reduction technique is carried out by dividing the final number by 10, and taking the remainder ("modulo 10"). The remainder of 23 divided by 10 is 3. Therefore, in 158711 is assigned

to bin 3. In this technique, the specific hash function is:

add all the digits;

take the remainder when divided by 10.

[0075] The same hash function can be used to convert any string into a number between 0 and 9. Different numbers can be used to find different hashes.

[0076] The hash function is repeatable, that is, any time the hash function receives the number 158711, it will always hash to bin 3. However, other numbers will also hash to bin 3. Any undesired string in the same bin as a desired string is called a hash collision.

[0077] Many other hash functions are known, and can be used. These include Cyclic Redundancy Checks (CRCs) commonly used for detecting errors in packet data in networks, a hash functions based on computing multiples of the data after division by a pre-specified modulus, the so-called Carter-Wegman universal hash functions (the simplest instantiation of which is to multiply the bit string by a suitably chosen matrix of bits), hash functions such as Rabin hash functions based on polynomial evaluation, and one-way hash functions such as MD-5 used in security. This list is not exhaustive and it will be understood that other

hash functions and other data reduction techniques can be used.

[0078] A specific aspect of the windowing embodiment (510) allows adding a part of the hash and removing a part when moving between two adjacent substrings. One aspect of this embodiment, therefore, may use an incremental hash function. Incremental hash functions make it easy to compute the hash of the next substring based on the hash of the previous substring. One classic incremental hash function is a Rabin hash function (used previously by Manber in spotting similarities in files instead of other non-incremental hashes (e.g, SHA, MD5, CRC32).

[0079] Large payloads may contain thousands of bytes, and according to one aspect, the content is data reduced at 300. 64-bit content may be sufficient to minimize the probability of hash collisions, so the data reduction may be, for example, a hash to 64 bits.

[0080] The string *S* may include information about the destination port. The destination port generally remains the same for a worm, and may distinguish frequent email content from frequent Web content or peer-to-peer traffic in which the destination port changes.

[0081] Figure 3 shows two different alternatives for detecting the frequent content. In the simple alternative,

shown as 305, a signature table is maintained. At 310, the string *S* is looked up in the signature table. If *S* is not in the table, it is added to the table at 315, along with a count of 0. However, if *S* is in the table, then the count associated with *S* is incremented at 320.

[0082] A frequency threshold is also defined. If the count for *S* exceeds the frequency threshold, then *S* is added to the frequent content table at 325. Periodically, each of the counters is reset, so that the frequent content test effectively requires the frequent content to be received in a specified time period.

[0083] The table formed in the simple solution 305 may be huge, and therefore a scalable solution is shown as 350. According to the scalable solution, a data reduction hash is first carried out.

[0084] An optional front end test that is commonly used is described in 351. One technique is to use a Bloom Filter as described in "Burton Bloom: Space/time tradeoffs in hash coding with allowable errors. Communications ACM, 1970" or a counting Bloom Filter as described in L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable wide-area Web cache sharing protocol SIGCOMM 98, 1998, to sieve out content that is repeated only a small number of times. While Bloom filters are a reasonable

front end and may suffice in some applications where memory is not expensive, they still require a few bits per piece of unique content and hence are not completely scalable.

[0085] For a more scalable analysis of frequent content, at 355, a number *k* of hash stages are established. Each stage *I* hashes the value *S* using a specified hash function *Hash(I)*, where Hash(I) is a different hash function for each stage I. For each of those stages, a specific position, *k*(I) is obtained from the hashing. The counter in position *k*(I) is incremented in each of the *k* stages. Then, the next I is established. Again, there are *k* stages, where *k* is often at least three, but could even be 1 or 2 in some instances.

[0086] At 360, the system checks to see if all of the *k* stage counters that incremented by the hash, for a specific string *S*, are greater than a stage frequency threshold. *S* is added to the frequent content table only when all of the *k* counters are all greater then the threshold.

[0087] Figure 4 illustrates the scalable technique 350 for k=3; a 3 stage hash. Each stage is a table of counters which is indexed by the associated hash function (Hash(I)) that is computed based on the packet content.

[0088] At the beginning of each measurement interval, all counters in each stage are initialized to 0. Each packet comes in and is hashed by a hash function associated with the stage. The result of the hash is used to set a counter in that stage. For example, the packet *S* is hashed by hash function 401 in order for the first stage 400. This produces a result of 2, shown incrementing counter 2 in the first stage array 400. The same packet S is also hashed using hash function 2 (421), and increments counter 0 in the second stage array 420. Similarly, and analogously, the packet *S* is hashed by hash function three (431), which hashes to counter 6 of the third stage 430. The same packet hashes to three different sections (in general, though there is a small probability that these sections may coincide) in the three different counter stages.

[0089] The stage detector 410 detects if the counters that have currently been incremented are each above the frequency threshold. The signature is added to the frequent content memory 415 only when all of the stages that have been incremented above the stage frequency threshold.

[0090] As examples, the first hash function 301 could sum digits and take the remainder when divided by 13. The second hash function 321 could sum digits and take the

remainder when divided by 37. Hash function 331 could also similarly be a third independent function. In practice, it parameterized hash functions may be used, with different parameters for the different stages, to produce different but independent instances of the hash function.

[0091] The use of multiple hash stages with independent hash functions reduces the problems caused by multiple hash collisions. Moreover, the system is entirely scalable. By simply adding another stage, the effect of hash collisions is geometrically reduced. Moreover, since the memory accesses can be performed in parallel, this can form a very efficient, multithreaded software or hardware implementation.

[0092] ***Spreading Content Test:***

[0093] Once a frequent signature is found at 215, 230 investigates whether the content is spreading characteristic. This is done by looking for and counting, sources and destinations associated with the content.

[0094] Figure 6a shows a flowchart of a simplistic and brute force way of doing this. According to figure 6a, a table of all unique sources and all unique destinations is maintained. Each piece of content is investigated to determine its source and its destination. For each string *S*, a table of sources and a table of destinations are

maintained. Each unique source or destination may increment respective counters. These counters maintain a count of the number of unique sources and unique destinations.

[0095] When the same string *S* comes from the same source, the counter is not incremented. When that same string does come from a new source, the new source is added as an additional source and the unique source counter is incremented. The destinations are counted in an analogous way. The source table is used to prevent over-counting the number of sources. That is, if Sally continually sends the message "hi Joe", Sally does not get counted twice.

[0096] During a large attack, millions or billions of sources and destinations may be involved. Figure 6b describes a data reduction technique that may be used in place of a huge table of unique sources and destinations.

[0097] First, at 629, an N bit count array is maintained where each bit may be 0 or 1, and the array has N positions, 630, the string *S* is reviewed against the frequent content table. If *S* is not within the table, no further steps are taken.

[0098] At 635, the source IP address is hashed, using a specified function, to form a W bit hash value S_{hash}. At 640, the hash value S_{HASH} is used to "set" the bit at a

corresponding bit position within the count array, if not already set. A running counter at 645 can maintain the number of set bits, or alternatively, the number of set bits can be counted at the end of each interval. 646 tests for the end of an interval, and resets the bits of the counter when the interval end is detected at 647.

[0099] The count of bits can be weighted by the probability of hash collisions, in order to get a more accurate count of numbers of sources and destinations.

[00100] This technique essentially stores a bitmap; one bit per source, instead of the entire 32-bit IP source table.

[00101] While the bitmap solution is better than storing complete addresses, it still may require keeping hundreds of thousands of bits per frequent content. Another solution carries out even further data compression by using a threshold T which defines a large value. For example, defining T as 100, this system only detects values that are large in terms of sources. Therefore, no table entries are necessary until more than 100 sources are found.

[00102] It also may be desirable to know not only the number of sources, but the rate of increase of the sources. For example, it may be desirable to know that even though a

trigger after 100 sources is made, that in the next second there are 200 sources, in the second after that there are 400 sources, and the like.

[00103] Figure 6c shows a system which allows scaling, e.g., by a scale factor, during the counting.

[00104] This system uses only a small portion of the entire bit map space.

[00105] At 650, a check is made to see if the signature is a frequent signature, if not, it skips the remaining steps. At 655, the IP address is hashed to a W bit number S_{HASH}. At 660, only certain bits of that hash are selected, e.g. the low order r bits. That is, this system scales down the count to only sampling a small portion of the bitmap space. However, the same scaling is used to *estimate* the complete bitmap space.

[00106] The same operations are carried out on the destination address at 605 and 670.

[00107] For example, an array of 32-bit (i.e., r = 32) may be maintained, where the threshold T is 96. Each source of the content is hashed to a position between 1 and 96. If the position is between 1 and 32, then it is set. If the position is beyond 32, then it is ignored, since there is no portion in the array for that bit.

[00108] At the end of the interval at 675, the number of bits set into the 32-bit array is counted, and corrected for collisions. The value is scaled up based on the number of bits which were ignored. Thus, for any value of T, the number of bits set within the available portion of the registers is counted, and scaled by a factor of T. *For example, in the previous example, if we had hashed from 1 to 96 but only stored 1 through 32, the final estimate would* be scaled up by a factor of 3.

[00109] Figure 6d illustrates a technique of using this scaling to count a rising infection over several intervals, by changing the scaling factor. A different scaling factor is stored along with the array in each interval. This technique can, therefore, reliably count from a very small to a very large number of sources with only a very small number of bits, and can also track rising infection levels.

[00110] The address is hashed at 682. A scale factor for sources is denoted by SourceScale. At 684 only if the high order bits of the hash from positions r+1 to r + SourceScale are all zero, is the low order r bits used to set the corresponding position in the Source BitMap. For example, if SourceScale is initially 3 and r is 32, essentially all but the low order 35 bits of the hash are

ignored and the low order 32 bits of the 35 bits are focused on, a scaling of 2^(35-32) = 2^3 = 8.

[00111] The interval is over at 686. At 688, the counter is cleared, and source scale is incremented by some amount. If in the next interval, in the same example, the scale factor goes up to 4, the scaling focuses on the top 36 bits of the hash, giving a scaling of 2^4 = 16. Thus by incrementing SourceScale by 1, the amount of sources that can be counted is doubled. Thus when comparing against the threshold for sources in 675, the number of bits in SourceHash is scaled by a factor of 2^(SourceScale - 1) before being compared to the threshold.

[00112] Note that this same technique is used not only for sources, as described, but also for destinations in 665, 670 and 675.

[00113] ***Scanning Test:*** A special kind of scanning test can also be carried out at 245. Unlike previous scanning systems, here both the content and the source are used as keys for the test, as compared with previous systems that tested merely the source. Tests are made for content that is being sent to unused addresses (of sources that disburse such content and send to unused addresses) and not solely sources. A guilt score is assigned to pieces of "bad" content, though as a side-effect, the individual stations

disbursing the bad content may be tagged. Notice also that the exploit in a TCP-based worm will not be sent to these addresses because a connection cannot be initiated without an answer from the victim

[00114] Noticing a range of probes to an unused space: A source may make several attempts to an inactive address or port by mistake. A hundred attempts to a single unused address or port is less suspicious than a single attempt to each of a hundred unused addresses/ports. Thus rather than counting just counting the number of attempts to unused addresses, it may also be useful to get an estimate of the range of unused addresses that have been probed.

[00115] To implement these augmentations scalably, a representation of the set of the unused addresses/ports of an enterprise or campus network is maintained. For scalability, unused addresses can be done compactly using a bitmap (for example, for a Class B network, 64K bits suffices) or Bloom Filter or as described in L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable wide-area Web cache sharing protocol SIGCOMM 98, 1998. The list can be dynamically validated. Initial guesses about which addresses spaces are being used can be supplied by a manager. This can easily be dynamically corrected. For example, whenever an address S thought to be unassigned

sends a packet from the inside, that address should be updated to be an assigned address. Note that in the special case of a contiguous address space, a simple network mask suffices.

[00116] A scalable list of unused ports can be kept by keeping an array with one counter for each port, where each array entry is a counter. The counter is incremented for every TCP SYN sent or each RESET sent, and decremented for every TCP FIN or FIN-ACK sent. Thus, if a TCP-based attack occurs to a port and many of the machines it contacts are not using this port, TCP FINs will not be sent back by these machines, or they will send TCP resets. Thus, the counter for that port will increase. Some care must be taken in implementing this technique to handle spoofing and asymmetrical routing, but even the simplest instance of this method will work well for most organizations.

[00117] A "blacklist" of sources that have sent packets to the unused addresses or ports in the last k measurement periods. This can be done compactly via a Bloom Filter or a bitmap. A hashed bit map can also be maintained, (similar to counting sources above) of the inactive destinations probed, and the ports for which scanning activity is indicated.

[00118] For each piece of frequent content, the mechanism keeps track of the range of sources in the blacklisted list associated with the content. Once again, this can be done scalably using a hashed bitmap as described herein.

[00119] Figure 7 shows testing for content of scanning. Again, this assumes a suspicious signature *S* at 700, the source address is hashed into a position S within the table. At 710, when the number of bits set within that suspicion table exceeds a threshold, then the scanning is reported as true.

[00120] ***Code Test:***

[00121] Code can also be detected at 245. This can be used with a packet code test shown in figure 8a. Within the suspicious sample, at each offset, a code test is run at 800. When code is detected to be over a specified length, the code test is reported to be positive at 805.

[00122] The code test can simply be a disassembler run on the packet at each of the plurality of offsets. Most worms and the like use 8086 code segments.Therefore, an 8086 disassembler can be used for this purpose.

[00123] Figure 8b shows an alternative technique of looking for opcodes and associated information associated with the opcodes. The opcodes may be quite dense, leaving only a few codes that are clearly not 8086 codes. Each opcode may

have associated special information following the code itself. While a small amount of data may look like code, because of the denseness of the opcodes, it is quite unlikely that large strings of random data look like codes. For example, if 90% of the opcodes are assigned, a random byte of data has a 90% chance of being mistaken for a valid opcode; however, this is unlikely to keep happening when measured over say 40 bytes of data that each of the appropriate bytes looks like a valid opcode.

[00124] This test, therefore, maintains a small table of all opcodes, and for each valid opcode, uses the length of the instruction to test whether the bits are valid. Figure 8b shows starting at Offset 0, doing a length test, at 850, and repeating until length greater than N for opcodes tests of length N. At 860, each bit at offset 0 along with its length in the opcode table, is looked up. If the opcode table indicates that the byte is invalid, the code test is indicated as failed, and exits at 865. If the opcode table entry is valid, the length test is incremented by the opcode table entry length value at 870, and the process continues.

[00125] The system thus checks for code at offset 0 by consulting the table looking for a first opcode at 0. If

the opcode is invalid, then the test fails, and the pointer moves to test the next offset.

[00126] However, if the opcode is valid, then the test skips the number of bytes indicated by the instruction length, to find the next opcode, and the test repeats. If the test has not failed after reaching N bytes from the offset O, then the test has succeeded.

[00127] This test can be carried out on each string, using only 8086 and unicode, since most of the attacks have been written in these formats. It should be understood, however, that this may be extended to other code sets, should that be necessary.

[00128] The code test can be combined with the frequent content test as a final confirmatory test of whether a piece of frequent content contains at least one fragment of code. In another alternative, the code detection can be used to form a front end for the frequent content. Only content that has a code segment of size N or more is therefore considered for frequent content testing.

[00129] ***Correlation Test:***

[00130] The following correlation test can be used in 245 to scalably detect the correlation between content sent to stations in one interval, and content sent by these sources in the next interval. As this is a likely sign of an

infection, it adds to the guilt score assigned to a piece of content, if this test is passed. A scalable method of doing this is shown in Figure 9.

[00131] In 905, a bitmap for sources and a bitmap for destinations are initialized to "0" whenever a new signature is added to the frequent content table. A similar initialization occurs at the end of every interval as shown in 940. The concepts used are very similar to those for detecting spreading content in Figure 6 and similar bitmaps are used. To distinguish them, these bitmaps are called SourceCorBitMap and DstCorBitMap instead of SrcBitMap and DstBitMap as in Figure 6.

[00132] In an analogous way for Figure 6, S is checked against a frequent content table at 910 in that a sample of sources are kept track as hashed bits in the source bitmap (915, 920) and a similar sample is kept track for destinations (925, 930). However, there is preferably no scale factor in the correlation test in Figure 9.

[00133] A test for correlation at 935 compares the bit positions set in the source bitmap for this interval with the bit positions set in the destination bitmap for the *previous* interval. If a large number of bits are set in common, it is clear that of the sample of destinations that received the content in the last interval, a significant

fraction is sending the same content in this interval, and the content passes the correlation test in 935.

[00134] The source and destination bitmaps of an interval are logged in 940 at the end of an interval and the counters are reset.

[00135] Another one of the additional checks shown as 245 is the Spam test shown in figure 10. An additional check carried out during the Spam test is to pass a so-called Bayesian Spam test of conventional type. The Bayesian test may heuristically analyze the content to determine if the suspected content is in fact Spam according to the Bayesian rules.

[00136] The packet testing described in Figure 2 can be carried out at any vantage links, e.g., where the enterprise communicates with the ISPs. The vantage links typically include a router, and according to the present system, a large-scale intrusion detection prevention device of the type disclosed herein operates within that router.

[00137] Two different sample combinations of the way that this can be used in a network configuration are respectively shown in figures 11A and 11B. Figure 11A shows in-line configuration systems 1100 and 1102 connected on respective sides of a router 1104. As an alternative or additional operation, a tap Configuration detection system

1110 may be used. The in-line system may have a function of actually blocking desired signatures, since it is actually in-line between the vantage link and the computer. The tap detection system does not actually block anything, but monitors the network contents and provides a signature as 1112 to the router 1104. The signature 1112 represents content that should be blocked by the router. All of these are in contact with the vantage link shown as 1099, which actually represents any connection to a public Internet of any sort.

[00138] Figure 11b shows an alternative connection in which the signatures are collected and sent to a central consolidator for processing. An advantage of this system is that different network portions from different locations can be collected, thereby providing a better sample of total network traffic. The private network 1150 is shown with a router 1152 guarding the entrance to that private network. According to this system, a number of different detection systems 1160, 1161, and 1162 each are located in either in-line or tap configurations on various networks or network points. For example, the different systems 1160, 1161 and 1162 may guard the entrance to different network facilities. Each of these operates according to the

systems described above, to create signatures of data portions as in 205.

[00139] The signatures from each of these devices, representing either the data, or the reduced data, is sent to a consolidator 1170. The consolidator then carries out the additional tests of finding the frequent content 215; finding spreading content 230, and additional tests 245. This has the advantage that a piece of content, while not passing the tests at any one device, still passes the tests when viewed across all the devices. Alternately, signatures 1164 or from device 1160, signatures 1165 from device 1161, and signatures 1166 from device 1162. The consolidated signature 1171 provides a consolidated signature the blocking to router 1152. More generally, however, any part of the intrusion detection system/intrusion prevention system, may generate signatures, that is bit patterns corresponding to the offensive content of the attack or undesired content, and the entire solution could distributed in various ways.

[00140] Other implementations, besides those specifically disclosed above, are within the scope of the following claims.

[00141] For example, although the above has described hashing, any technique, mathematical or otherwise, can be

used for this data reduction so long as it follows the techniques disclosed above. Moreover, although the above has described only a few tests, which are representative of today's kinds of intrusions, other tests can be added in the future.

[00142] All such implementations and modifications are intended to be encompassed within the following claims.